Contenido

10.

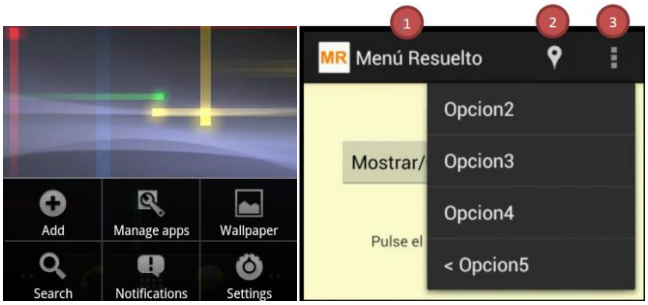
Menús

INTRODUCCIÓN. ActionBar ó AppBar

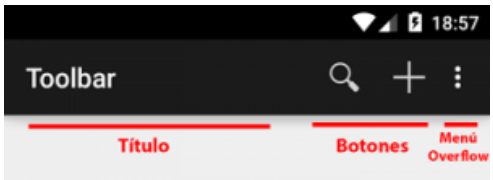
Los menús forman parte de la interfaz de usuario. Son la parte común entre todas las aplicaciones ya que nos permite pasar de una actividad a otra cómodamente y ejecutar acciones seleccionando una opción del mismo.

A partir de Android 3.0 (API 11) Google cambió el concepto de menú que venía utilizando: en lugar de un panel en la parte inferior con hasta 6 ítems, decidió eliminar dicho menú (para que no hiciera falta usar el botón menú de los dispositivos móviles) y utilizar la Action Bar para mostrar las opciones.

La *Action Bar* **(1)** es la barra que hay en la parte superior de toda aplicación y se suele componer de un icono junto con el nombre de la aplicación, y los distintos ítems del menú, que pueden estar visibles **(2)** o agrupados en el *Overflow Menu Button* **(3)** (botón que despliega un menú con el resto de opciones).



Con la llegada de **Material Design y Android 5.0** la *Action bar*, **o *Appbar* como se la ha rebautizado**, sigue apareciendo en la parte superior y tiene un aspecto muy parecido a la de versiones anteriores. Una de las principales modificaciones es que desde la llegada de Material Design, ya no se recomienda que el icono de la aplicación aparezca a la izquierda del título. Lo que sí puede aparecer en ese lugar son los iconos indicativos de la existencia de menú lateral deslizante (*navigation drawer*) o el botón de navegación hacia atrás/arriba, pero esto ya lo veremos más adelante.



En la actualidad, Android proporciona este componente a través de la librería de soporte appcompat-v7, que podemos incluir en nuestro proyecto añadiendo su referencia. De

cualquier forma, en versiones actuales de Android Studio, esta referencia viene incluida por defecto al crear un nuevo proyecto en blanco.

Otro punto importante y que puede dar problemas a la hora de reutilizar código creado con versiones anteriores, es que el tema utilizado por nuestra aplicación debe ser uno de los proporcionados por la librería appcompat-v7. Para ello abrimos el fichero styles.xml situado en la carpeta /res/values y nos aseguramos que el tema de nuestra aplicación extiende de alguno de los temas Theme.AppCompat disponibles, por ejemplo como se puede ver abajo Theme.AppCompat.Light.DarkActionBar (fondo claro con action bar oscura):

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

    </style>
</resources>
```

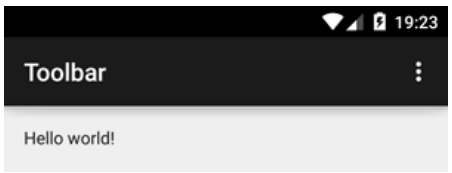
Menús

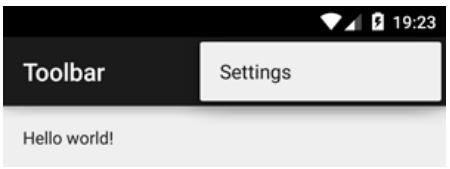
Hay 4 tipos de menú en Android:

1. *Options Menu y Action Bar*: Menú de opciones y barra de acción.

Se corresponde con los menús mostrados en la ActionBar de los que hemos hablado antes y que se despliegan con el botón de overflow. 

Este menú se utiliza para mostrar las opciones básicas de la aplicación y puede ser común a varias *activities* de la misma APP.





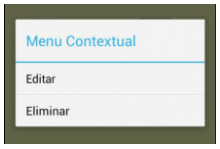
¡¡IMPORTANTE!! Si el dispositivo tiene un botón de menú, el icono de overflow no se mostrará. Podremos acceder al menú de la Action Bar desde el botón del dispositivo.

2. *Contextual Menus:* Menús contextuales.

Son los menús que aportan opciones extra para un determinado elemento. Se suele utilizar con las *ListView, RecyclerView, GridView* u otras colecciones, para que al hacer una pulsación larga sobre un determinado elemento de la lista nos permita realizar acciones únicas y específicas sobre él, como por ejemplo editar el elemento, borrarlo, etc.

Hay 2 tipos de menús contextuales:

2.1. *Floating Context Menu*: Menú contextual flotante.



Este menú abre un menú flotante que se superpone a la *activity*. Si se hace click fuera del menú, éste se cerrará.

2.2. *Contextual Action Mode*: Modo de acción contextual.

Menú contextual recomendado a partir de la API 11.

Al abrir este menú realizando una pulsación larga sobre el elemento se abre una nueva barra de acción llamada *Contextual Action Bar (CAB),* que se superpone encima de la barra normal de la aplicación y añade una serie opciones para el/los ítem(s) seleccionados, puesto que este tipo de menú contextual permite la selección múltiple.

Es muy útil, por ejemplo, cuando se quiere borrar varios elementos de una lista de manera rápida y cómoda.

3. Popup Menú: Menú emergente.

Actúa muy similar al *Overflow Menu Button*, pero éste se encuentra dentro de la aplicación y no en la *Action Bar*. Al pulsar sobre el menú se muestra una lista vertical de opciones.

Se usa para cuando se quiere realizar cambios en una determinada zona de la *activity*. Las acciones del menú emergente no deben afectar directamente al contenido, ya que esa es la misión de los menús contextuales.

Este tipo de menús desplegables solo está disponible para la API 11 y superior.

4. Menú lateral deslizante o *Navigation Drawer*: este tipo de menús de navegación ya llevaban tiempo utilizándose y proliferando por el *market* en numerosas aplicaciones, pero igual que pasaba con la *action bar* en versiones de *Android* anteriores a la 3.0, se hacía gracias a librerías externas que implementaban este componente. Google ha querido acabar con esto aportando su propia implementación de este componente y definiendo su comportamiento en las guías de diseño de la plataforma.

El *navigation drawer* está disponible como parte de la librería de compatibilidad *android-support*. Para poder utilizarlo en nuestras aplicaciones tendremos que asegurarnos que tenemos incluida en nuestro proyecto la librería *android-support-v4.jar* (debe ser la revisión 18 o superior.

El usuario puede hacer aparecer este menú realizando un deslizamiento desde el borde de la parte derecha o izquierda de la pantalla o bien pulsando el icono de aplicación de la *Action Bar*.

Definición de un menú XML

Para crear un menú en nuestra aplicación, independientemente del tipo que sea, debemos crear un archivo xml en la ruta /res/menu. Por defecto, al crear un proyecto de Android, se

crea el menú main.xml, podemos modificar éste o crear uno nuevo con un nombre que nos ayude a identificarlo mejor, como por ejemplo menú_principal.xml.

La estructura xml básica de un menú es la siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_save"
          android:icon="@drawable/menu_save"
          android:title="@string/menu_save" />
    <!-- menu group -->
    <group android:id="@+id/group_delete">
        <item android:id="@+id/menu_archive"
              android:title="@string/menu_archive" />
        <item android:id="@+id/menu_delete"
              android:title="@string/menu_delete" />
    </group>
</menu>
```

<menu>...</menú>

Define un menú y por tanto, contendrá las opciones del menú. El elemento **<menu>** debe ser la raíz del documento *xml* y sólo podrá contener uno o más elementos **<ítem>** y <**group>.**

<**ítem>…</ítem>**

Representa a un único ítem del menú (una opción). Este elemento puede contener a un único elemento **<menu>** para crear un submenú.

Atributos:

- android:id -> ID único que representa al ítem y nos permite enlazarlo en la activity.

- android:icon -> Icono para la opción del menú.

- android:title -> Texto de la opción del menú.

-android:showAsAction -> Especifica si debe aparecer en la ActionBar o encontrarse escondido en el Overflow Menu Button.

Sus parámetros son:

- **"ifRoom":** Coloca el icono del ítem en la Action Bar únicamente si cabe (por espacio).
- **"withText":** Se debe usar junto a "ifRoom" o "always" y mostrara el "icon" + "title" si tiene espacio.
- **"never":** Nunca se muestra en la Action Bar, por tanto el ítem estará en la lista desplegable del Overflow Menu Button.
- **"always":** Siempre se muestra en la Action Bar. No se debe usar este parámetro pues ocasiona problemas si realmente no tiene espacio.
- **"collapseActionView":** Indica que la View asociada a éste ítem es de tipo plegable (para menús desplegables a la izquierda).

<group>…<group>

Opcional. Sirve para agrupar elementos **<ítem>** que compartan propiedades como si están activos, o se deben mostrar u ocultar, etc. No tiene ninguna repercusión visual a la hora de mostrar el menú.

Atributos:

- android:id -> ID único que representa grupo de items.

- android:checkableBehavior -> Forma de comportarse con respecto al hacer click en algún ítem del grupo.

Sus parámetros son:

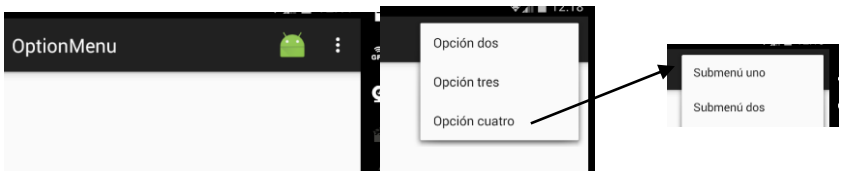
- **"none":** No se pueden seleccionar los ítems del grupo.
- **"all":** Todos se pueden seleccionar. Coloca un checkbox en cada ítem.
- **"single":** Únicamente se puede seleccionar un ítem. Coloca un radio button en cada ítem.
- **android:visible ->** "true" si queremos que se muestre el grupo de ítems, "false" si no.
- **android:enabled ->** "true" para habilitar el grupo de ítems, "false" si no.

Referencia completa:

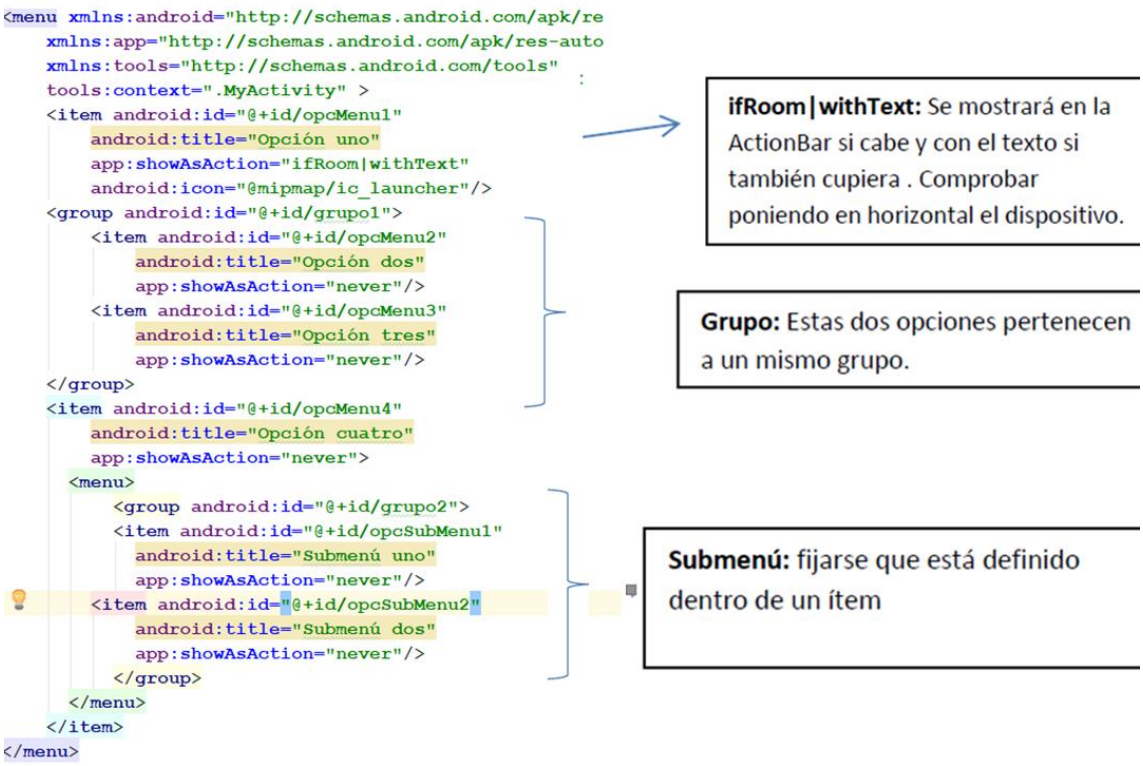
http://developer.android.com/intl/es/guide/topics/resources/menu-resource.html

Definición de un menú de opciones

Vamos a crear un sencillo menú de opciones en la Action Bar. El menú tendrá 4 opciones, la primera será una acción (se llama acciones a las opciones del menú que están visibles en la Action Bar y no ocultas en el Overflow Menu Button), dos opciones pertenecerán a un mismo grupo y estarán dentro del Overflow Menu Button, y la última opción también lo estará pero además tendrá un submenú con dos ítems más. Al pulsar sobre cualquier opción mostraremos cuál se ha pulsado con un Toast.



La estructura *xml* del menú **/res/menu/main.xml** será así:




```
<menu xmlns:android="http://schemas.android.com/apk/re
    xmlns:app="http://schemas.android.com/apk/res-auto
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MyActivity" >
    <item android:id="@+id/opcMenu1"
        android:title="Opción uno"
        app:showAsAction="ifRoom|withText"
        android:icon="@mipmap/ic_launcher"/>
    <group android:id="@+id/grupo1">
        <item android:id="@+id/opcMenu2"
            android:title="Opción dos"
            app:showAsAction="never"/>
        <item android:id="@+id/opcMenu3"
            android:title="Opción tres"
            app:showAsAction="never"/>
    </group>
    <item android:id="@+id/opcMenu4"
        android:title="Opción cuatro"
        app:showAsAction="never">
      <menu>
          <group android:id="@+id/grupo2">
          <item android:id="@+id/opcSubMenu1"
            android:title="Submenú uno"
            app:showAsAction="never"/>
        <item android:id="@+id/opcSubMenu2"
            android:title="Submenú dos"
            app:showAsAction="never"/>
          </group>
      </menu>
    </item>
</menu>
```

Una vez tengamos el archivo xml con la estructura del menú, crear un menú de tipo “menú de opciones” es sencillo.

En el MainActivity.java o en cualquier otra activity, se encuentra el método onCreateOptionsMenu():

```
public class MainActivity extends AppCompatActivity {

    //...

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
}
```

Para la versión de Android 2.3.x (API 9) y anteriores, este método es llamado al pulsar por primera vez el botón menú del dispositivo, para las siguientes versiones de Android, se llama nada más comience la activity.

En éste método inflaremos el menú con getMenuInflater().inflate(…) y le pasaremos la id de nuestro menú xml creado, en este caso: R.menu.activity_main (aunque podría ser cualquier otro), y el objeto menu. De esta forma habremos asignado el xml del menú a la aplicación para que lo muestre.

Ahora queremos controlar qué hacer cuando se seleccione un ítem, para ello debemos añadir a nuestro activity el método onOptionsItemSelected():

```
public boolean onOptionsItemSelected(MenuItem item) {
    String texto = "";

    switch(item.getItemId()) {
        case R.id.opcMenu1:
            texto = "1";
            break;
        case R.id.opcMenu2:
            texto = "2";
            break;
        case R.id.opcMenu3:
            texto = "3";
            break;
        case R.id.opcMenu4:
            texto = "4";
            break;
        case R.id.opcSubMenu1:
            texto = "4, el submenú 1";
            break;
        case R.id.opcSubMenu2:
            texto = "4, el submenú 2";
            break;
        default:
            return super.onOptionsItemSelected(item);
    }
    Toast.makeText(getApplicationContext(), "Ha pulsado el menú " + texto, Toast.LENGTH_SHORT)
        .show();
    return true;
}
```

Éste método es llamado cada vez que se pulse sobre un ítem del menú.

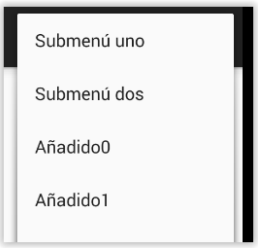
Nos proporciona un parámetro de tipo *MenuItem*, con lo que tendremos información para saber qué opciones del menú se ha pulsado comparando su ID, y podremos realizar una acción individual para cada elemento. Si se ha controlado la acción para el ítem, se debe devolver **true,** si no, se debe llamar a su constructor, (en el **default** del *switch*).

Por último, tal vez queramos realizar cambios en el menú durante la ejecución de la aplicación. Si quisiéramos esto, tendríamos que añadir el método *onPrepareOptionsMenu().*

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);

    menu.add(groupId, itemId, order, title);
    menu.add(IDGrupo, IDItem, OrdenEnMenu, TextoItem);

    return true;
}
```

Este método es llamado cada vez que se muestra el menú, por lo tanto nos permite realizar modificaciones dinámicamente. Con el método *menu.add(…)* añade nuevos elementos al menú. Por último, debemos devolver **true** si queremos que se muestre el menú de opciones.

Crea una app que muestre un menú de opciones siguiendo el ejemplo anterior. Tendrá que agregar un elemento AñadidoX(x se incrementará) al submenú del elemento4, cada vez que se muestre el menú. Al seleccionar un elemento del menú se mostrará un Toast con el texto del ítem pulsado.



Definición de un menú contextual flotante

Primero explicaremos el menú contextual flotante, el otro tipo de menú contextual (el modo de acción contextual) se explica más adelante.

Lo que queremos es que al pulsar durante un tiempo largo sobre un control, ya sea de tipo *TextView*, *Button*, un ítem de un RecyclerView, *ListView* o de un *GridView*, etc., se abra un menú flotante con unas opciones que afecten únicamente a él.

Para asignar un menú contextual a un elemento, lo primero que tenemos que hacer es registrarlo con el método *registerForContextMenu().* Esto lo haremos en el método *onCreate()* de nuestra *activity*.

Por ejemplo, si tenemos el *TextView labelHello*, basta con llamar a

```
labelHello=(TextView) findViewById(R.id.label1);
registerForContextMenu(labelHello);
```

Para que el sistema sepa que sobre ese TextView se debe incorporar un menú contextual.

Acto seguido, se implementará el método onCreateContextMenu(), que deberemos definir en el ActivityMain.java:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo){
    super.onCreateContextMenu(menu, v, menuInfo);

    switch (v.getId()){
    case R.id.label1:
        MenuInflater inflater=getMenuInflater();
        inflater.inflate(R.menu.menu_contextual, menu);
        menu.setHeaderTitle("Menu Contextual");
    }
}
```

Aquí lo que hacemos es crear un menú contextual propio (*menu_contextual.xml*) para el *TextView labelHello*.

Si se quisiera crear distintos menús contextuales para cada elemento, bastaría con añadir más cases al *switch* con la id de cada elemento.

Con el *MenuInflater* inflamos el menú y le asignamos nuestro menú personalizado que ya deberíamos tener en la ruta */res/menú.*

Por ejemplo, en nuestro caso:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/opmenu1"
        android:title="Editar"
        android:orderInCategory="100"
        android:showAsAction="never" />

     <item android:id="@+id/opmenu2"
        android:title="Eliminar"
        android:orderInCategory="100"
        android:showAsAction="never" />
</menu>
```

Por último le hemos añadido un título al menú flotante con *menú.setHeaderTitle("Menu Contextual").*

MenúContextual y RecyclerView → En caso de querer insertar un menú contextual al pulsar un elemento del RecyclerView, se deberá salir un poco del anterior esquema. En primera instancia derivaremos el Holder de ***View.OnCreateContextMenuListener.***

```
public class Holder extends RecyclerView.ViewHolder implements View.OnCreateContextMenuListener {
    TextView textView;
    public Holder(View itemView) {
        super(itemView);
       textView=(TextView)itemView.findViewById(R.id.textView);
        itemView.setOnCreateContextMenuListener(this);

    }
    public  void Bind(String item)
    {
      textView.setText(item);
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {
        MenuInflater menuInflater=new MenuInflater(v.getContext());
        menuInflater.inflate(R.menu.menucontextual_texto, menu);

    }
```

Como se puede ver en la imagen, se aplica el Listener del menú contextual al itemView del Holder y sobreescribimos el onCreateContextMenu para inflar el menú contextual.

Para capturar sobre que elemento del menú contextual flotante se ha pulsado sobrescribiremos el método *onContextItemSelected(), de la activity:*

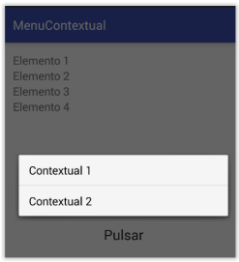
```
@Override
public boolean onContextItemSelected(MenuItem item){
    String s="";

    switch (item.getItemId()){
        case R.id.opmenu1:
            s="opcion1 del menu contextual";
            break;
        case R.id.opmenu2:
            s="opcion2 del menu contextual";
            break;
    }
    Toast.makeText(getApplicationContext(),s,Toast.LENGTH_LONG).show();
    return true;
}
```

Éste método es llamado cuando se haga un click sobre una opción de este menú.

Como nos proporciona el *MenuItem* ítem, podemos saber a partir de su ID de que opción en concreto se trata y ejecutar el código que queramos que realice.

Al igual que en el método similar del menú de opciones, se devolverá **true** si se ha tratado el ítem seleccionado, y en caso de que no se trate (**default**), se llamará al constructor del método *super.onContextItemSelected(ítem).*

Crea una app que muestre una lista creada con RecyclerView y un texto (pulsar). Se deberá mostrar un menú contextual, tanto al pulsar el texto como al pulsar cualquiera de los elementos de la lista. Al seleccionar un elemento del menú se mostrará un Toast con el texto del ítem pulsado.



Definición de un menú popup

Al igual que en los demás, creamos primero su estructura con un archivo xml: *menú_contextual.xml* en la ruta *res/menú/.*

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/popmenu1"
        android:title="CambiarTexto"
    />

    <item android:id="@+id/popmenu2"
        android:title="CambiarColor"
    />
</menu>
```

Crearemos el menú emergente directamente en el método *onCreate***()** de nuestra *activity*.

Su estructura es similar a como cuando creábamos un *onClickListener* de un botón, ya que el popup se mostrará al hacer click sobre un botón, aunque se procedería de igual manera sobre cualquier otro elemento. En nuestro caso hemos introducido un botón en nuestro *layout* principal.

```
btnClic=(Button)findViewById(R.id.cmdMostrar);
btnClic.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
        PopupMenu popup=new PopupMenu(getApplicationContext(),v);
        popup.getMenuInflater().inflate(R.menu.menu_popup,popup.getMenu());

        popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {

            public boolean onMenuItemClick(MenuItem item) {
                String s="";

                switch (item.getItemId()){
                case R.id.popmenu1:
                    s="opcion1 del menu poup";
                    break;
                case R.id.popmenu2:
                    s="opcion2 del menu poup";
                    break;
                }
                Toast.makeText(getApplicationContext(),s,Toast.LENGTH_LONG).show();
                return true;
            }
        });
        popup.show();
    }
});
```

Creamos su *onClickListener* y dentro definimos un objeto de la clase *PopupMenu* pasándole el contexto y la View en la que se va a mostrar. *(new PopuMenu(getApplicationContext(), v).*

Después debemos inflar el *PopupMenu* donde le especificamos el xml que contiene la estructura del menú (*R.menu.menu_popup*) y como segundo parámetro, le pasamos el menú en el que se va a inflar (*popup.getMenu()*).

A continuación definimos un *Listener* para controlar la acción cuando se pulse sobre un ítem del menú emergente, esto lo hacemos con el *onMenuItemClick(MenuItem item).*

Es aquí donde a partir del ID del ítem (*ítem.getItemID()*) podemos definir qué hacer cuando se pulse en la opción del menú emergente indicada.

Por último, no debemos olvidar mostrar el menú con *popup.show().*

- **Implementa el ejemplo del popupMenu**

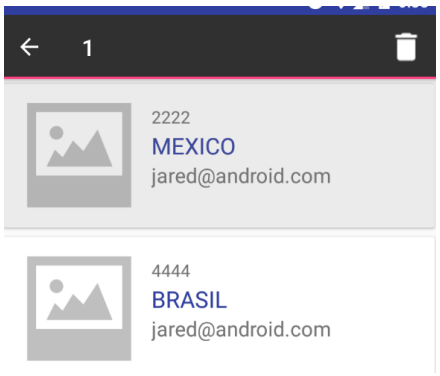
- **Ejercicio PropuestoMenús.**

Contextual Action Mode. Menú de selección múltiple

Es un modo de menú que tras su activación (tras una pulsación larga, siguiendo las guías de diseño de Android) permite actuar sobre diferentes elementos a los que luego se les aplica una opción del menú (que habrá cambiado al activar este modo).

Para crear un menú de selección múltiple en nuestro recycler deberemos de seguir una serie de pasos:

1. Activar la toolbar específica para el menú multiselección, que permita realizar las funciones necesarias sobre los elementos seleccionados. Esto lo haremos creando una clase derivada de ActionMode.Callback.
2. Crear una clase que permita gestionar los elementos seleccionados de la lista, para eso utilizaremos una clase abstracta derivada de la clase RecyclerView.Adapter. Utilizaremos un SparseBooleanArray para guardar las posiciones seleccionadas.
3. Crear un atributo booleano en el tipo que utilizamos para los datos. Este nos permitirá guardar si hemos seleccionado el elemento.
4. Controlar en el Holder el atributo de selección para producir el efecto deseado, que identifique el elemento pulsado (cambio de color de fondo, mostrar imagen, etc).
5. Gestionar en la clase MainActivity y en el Adaptador los métodos necesarios para activar la ActionMode, seleccionar los elementos y realizar las acciones del menú.



Vamos a explicar, más extensamente, cada uno de los pasos nombrados:

1. Activar toolbar mediante ActionMod

En la clase de la actividad donde vayamos a incluir el recycler, tendremos que crear una clase interna derivada de ActionMode.Callback con los métodos sobrescritos necesarios, esta clase será la que permita construir y manejar la toolbar.

```
private class ActionModeCallback implements ActionMode.Callback {
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        mode.getMenuInflater().inflate (R.menu.selected_menu, menu);
        return true;
    }
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) { return false; }

    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_remove:
                adaptador.eliminarItemsSeleccionados(adaptador.getSelectedItems());
                mode.finish();
                return true;
            default:
                return false;
        }
    }
    @Override
    public void onDestroyActionMode(ActionMode mode) {
        adaptador.clearSelection();
        adaptador.desactivarSelección();
        actionMode = null;
    }
}
```

En el código podemos ver el método onCreateActionMode, que estará encargado de inflar el xml del menú de la Toolbar. En el método onActionItemClicked será donde gestionemos lo que queremos que ocurra al pulsar un elemento del menú, en este caso sería eliminar el ítem seleccionado, no debemos olvidar mode.finish() para indicar que se debe cerrar la barra una vez termine la acción. El método onDestroyActionMode será llamado en el momento en que se va a eliminar la toolbar, por lo que debemos poner el objeto a null y limpiar la selección de ítems que hayan quedado pulsados.

2. Clase para gestionar los elementos seleccionados

Esta clase se encarga de controlar las posiciones de los elementos del recycler que han sido seleccionados o posteriormente deseleccionados, para ello utilizar un SparseBoolenArray. Este array se encarga de mapear los índices de los elementos seleccionados guardando el número de índice junto a un valor true en caso de estar seleccionado. Los métodos de la clase se explican por sí solos. Por otro lado esta clase puede ser utilizada tal y como viene en el código inferior sin necesidad de realizar ningún cambio.

```
import android.support.v7.widget.RecyclerView;
import android.util.SparseBooleanArray;
import java.util.ArrayList;
import java.util.List;

abstract class SeleccionableAdapter extends RecyclerView.Adapter{
    private SparseBooleanArray selectedItems;

    public SeleccionableAdapter() {
        selectedItems = new SparseBooleanArray();
    }
    boolean isSelected(int position) {
        return getSelectedItems().contains(position);
    }
    boolean toggleSelection(int position) {
```


```
        boolean seleccionado;
        if (selectedItems.get(position, false))
        {//devuelve true si encuentra en la posición indicada el campo correspondiente del
//sparceBoolean a true. Si no lo encuentra o está a false devolverá false
            selectedItems.delete(position);
            seleccionado=false;
        } else {
            selectedItems.put(position, true);
            seleccionado=true;
        }
        notifyItemChanged(position);
        return  seleccionado;
    }
    void clearSelection() {
        List<Integer> selection = getSelectedItems();
        selectedItems.clear();
        for (Integer i : selection) {
            notifyItemChanged(i);
        }
    }
    int getSelectedItemCount() {
        return selectedItems.size();
    }
    List<Integer> getSelectedItems() {
        List<Integer> items = new ArrayList<>(selectedItems.size());
        for (int i = 0; i < selectedItems.size(); ++i) {
            items.add(selectedItems.keyAt(i));
        }
        return items;
    }
}
```

3. Atributo Boolean en el tipo de los datos

Deberemos guardar en alguna variable el estado de la selección, una manera de hacerlo sería creando un atributo en el tipo para guardarlo.

```
class Usuario {
    private String nombre;
    private String apellidos;
    private String correo;
    private boolean seleccionado;

    Usuario(String nombre, String apellidos, String correo, boolean seleccionado) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.correo = correo;
        this.seleccionado=seleccionado;
    }
```

4. Mostrar el efecto deseado al seleccionar un elemento

Usando el método bind del Holder, preguntamos si el dato que se pasa como argumento tiene su atributo de selección a true y en ese caso haremos lo necesario para resaltar esa línea del recycler. En este caso mostraremos u ocultaremos una vista que teníamos en el layout de la línea del recycler.

```
void bind(Usuario usuario)
{
    txtNombre.setText(usuario.getNombre());
    txtApellido.setText(usuario.getApellidos());
    this.usuario =usuario;
    viewOverlay.setVisibility(usuario.isSeleccionado() ? View.VISIBLE : View.INVISIBLE);
}
```


En esta imagen podemos ver la View dentro del CardView, que en un principio está oculta y se le ha dado un color verde con transparencia. No está incluido todo el código del layout, por ser evidente:

```
        </LinearLayout>
            <ImageButton
                android:id="@+id/imageButton"
                android:backgroundTint="@android:color/transparent"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:src="@android:drawable/ic_menu_share" />
        </LinearLayout>
        <View
            android:id="@+id/selected_overlay"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@color/fondoCardViewPussada"
            android:visibility="invisible" />
    </android.support.v7.widget.CardView>
```

5. Gestionar la selección, activar la Toolbar y

```
@Override
public boolean onLongClick(View view) {
    int position=recycler.getChildAdapterPosition(view);
    if (actionMode == null) {
        actionMode = startSupportActionMode(actionModeCallback);

    }
    if(intercambiarSeleccion(position))
    {
        datos[position].setSeleccionado(true);
    }
    else   datos[position].setSeleccionado(false);

    return true;
}
```

```
private ActionModeCallback actionModeCallback
                    = new ActionModeCallback();
private ActionMode actionMode;
```

En el Click largo, será donde nos crearemos el objeto de tipo ActionMode en el caso en que todavía no estuviera activado. También se codificará la selección o deselección del elemento pulsado y actualizará a true o false el atributo correspondiente del dato.

En el método Click, el código será el mismo:

```
@Override
public void onClick(View view) {
    int position=recycler.getChildAdapterPosition(view);
    if (actionMode != null) {
        if(intercambiarSeleccion(position))
        {
            datos[position].setSeleccionado(true);
        }
        else   datos[position].setSeleccionado(false
        );
    }
   else  Toast.makeText(MainActivity.this,datos[position].getCorreo(),Toast.LENGTH_LONG).show();
}
```

El método intercambiarSelección al que se hace referencia en los dos anteriores métodos, llamará a su vez al método del adaptador toggleSelection.

```
private boolean intercambiarSeleccion(int position) {
    boolean seleccionado=adaptador.toggleSelection(position);
    int count = adaptador.getSelectedItemCount();

    if (count == 0) {
        actionMode.finish();
    } else {
        actionMode.setTitle(String.valueOf(count));
        actionMode.invalidate();
    }
    return  seleccionado;
}
```

La clase Adapter la derivaríamos de la clase abstracta seleccionableAdapter, los métodos serían los normales de esta clase y le añadiríamos los métodos del adaptador que nos permitirían gestionar la eliminación de elementos y la desactivación de todos los elementos seleccionados.

```
void eliminarItemsSeleccionados(List<Integer> items)
{
    Collections.sort(items);
    Collections.reverse(items);
    //Código para convertir el Array en lista y poder eliminar más facilmente
    //el elemento
    ArrayList<Usuario> aux=new ArrayList(Arrays.asList(((MainActivity)context).datos));
    for (int i:items) aux.remove(i);
    ((MainActivity) context).datos=new Usuario[aux.size()];
    ((MainActivity)context).datos=aux.toArray(((MainActivity)context).datos);
    for (int i:items) notifyItemRemoved(i);
}
void desactivarSelección()
{
    for (Usuario i:((MainActivity)context).datos) i.setSeleccionado(false);
}
```

Desarrolla el ejemplo del tema para practicar los menús contextuales o ActionMode. EjercicioResueltoMultiselección.

Menú Lateral. NavigationView

El menú lateral deslizante o Navigation View es el que aparece en muchas aplicaciones al deslizar el dedo desde el borde izquierdo de la pantalla hacia el lado opuesto (también puede aparecer en el lado derecho, pero es menos frecuente), o cuando pulsamos el icono que se coloca en la ToolBar.

Un elemento NavigationView se define dentro de un DrawerLayout. Este último elemento actúa como un contenedor de nivel superior. Y en casi todos los casos, deberemos crear una Toolbar como vimos en el tema3 (Tema en el que incluimos MaterialDesign). Por tanto deberemos incluir la librería de material design para poder incluir los elementos necesarios. Para poder manejar el menú lateral desde la ToolBar, deberemos incluir en la MainActivity el siguiente código, que nos permite añadir la ToolBar a la ActionBar y activar el botón Home:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
    setSupportActionBar(toolbar);
    ActionBar actionBar = getSupportActionBar();
    actionBar.setHomeAsUpIndicator(R.drawable.ic_menu);
    actionBar.setDisplayHomeAsUpEnabled(true);
```


```
<android.support.v4.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="false">
        <android.support.design.widget.AppBarLayout
            android:id="@+id/app_bar_layout"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:theme="@style/ThemeOverlay.AppCompat.Dark"
            android:gravity="center_horizontal">

                <android.support.v7.widget.Toolbar
                    android:id="@+id/toolbar"
                    android:layout_width="match_parent"
                    android:layout_height="?attr/actionBarSize"
                    android:background="?attr/colorPrimary"
                    app:layout_collapseMode="pin"/>
        </android.support.design.widget.AppBarLayout>
        <android.support.design.widget.NavigationView
            android:id="@+id/navigation_view"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            android:fitsSystemWindows="true"
            app:headerLayout="@layout/drawer_header"
            app:menu="@menu/drawer" />
</android.support.v4.widget.DrawerLayout>
```

Es importante colocar la propiedad del DrawerLayout fitsSystemWindows a true **android:fitsSystemWindows="true"** con el fin de que el menú aparezca por debajo de la StatusBar.

Dos elementos son importantes en la definición del NavigationView,

```
app:headerLayout="@layout/drawer_header"
app:menu="@menu/drawer"/>
```

Que definen el diseño de la cabecera del menú, y el layout del propio menú.

También hay que tener en cuenta la propiedad android:layout_gravity, que determina el lado de la pantalla por el que aparecerá el menú deslizante (“start” para que aparezca por la izquierda, o “end” por la derecha).

En primer lugar creamos el archivo drawer_header en la carpeta res/layout:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/background"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_gravity="center_vertical"
        android:scaleType="fitCenter"
        android:src="@drawable/imgfondo" />

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="16dp"
        android:theme="@style/ThemeOverlay.AppCompat.Dark"
        android:gravity="bottom">

        <TextView
            android:textColor="?attr/colorPrimaryDark"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Drawer Header"
            android:textAppearance="@style/TextAppearance.AppCompat.Body1"/>
    </LinearLayout>
</FrameLayout>
```


Para poner una imagen de fondo en la cabecera hay que incluir la definición anterior dentro de un FrameLayout que además contenga la imagen de fondo.

A continuación creamos el archivo drawer.xml en la carpeta res/menu

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <group android:checkableBehavior="single">
        <item
            android:id="@+id/navigation_item_attachment"
            android:checked="true"
            android:icon="@drawable/ic_attachment"
            android:title="@string/nav_item_attachment" />
        <item
            android:id="@+id/navigation_item_images"
            android:icon="@drawable/ic_image"
            android:title="@string/nav_item_images" />
        <item
            android:id="@+id/navigation_item_location"
            android:icon="@drawable/ic_place"
            android:title="@string/nav_item_location" />
    </group>

    <item android:title="@string/nav_sub_menu">
        <menu>
            <item
                android:icon="@drawable/ic_emoticon"
                android:title="@string/nav_sub_menu_item01" />
            <item
                android:icon="@drawable/ic_emoticon"
                android:title="@string/nav_sub_menu_item02" />
        </menu>
    </item>
</menu>
```

Creamos el menú, que tendrá dos apartados. En el primero aparecerá chequeada la opción del menú por defecto. La segunda viene separada por un subtítulo para separar los elementos del primer grupo de los del segundo.

Solamente nos falta programar en el archivo MainActivity.java la aparición de menú lateral y las distintas opciones del mismo.

En el método onCreate(), necesitamos la variable que referencie al DrawerLayout y lo asocie con nuestro elemento en la vista:

```
mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
```

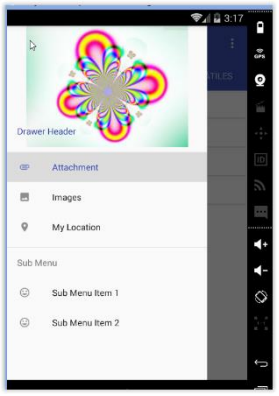
A continuación modificamos el método onOptionsItemSelected() de forma que cuando se pulse el icono de home aparezca el menú lateral (por defecto aparece al deslizar el dedo dese la parte izquierda de la pantalla).

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    switch (id) {
        case android.R.id.home:
            mDrawerLayout.openDrawer(GravityCompat.START);
            return true;
        case R.id.action_settings:
            return true;
    }

    return super.onOptionsItemSelected(item);
}
```

Para capturar las pulsaciones que se realizan sobre el menú lateral necesitamos establecer un onNavigationItemSelectedListener en el método onCreate().

```java
NavigationView navigationView = (NavigationView) findViewById(R.id.navigation_view);
navigationView.setNavigationItemSelectedListener(new NavigationView.OnNavigationItemSelectedListener() {
    @Override
    public boolean onNavigationItemSelected(MenuItem menuItem) {
        menuItem.setChecked(true);
        mDrawerLayout.closeDrawers();
        Toast.makeText(MainActivity.this, menuItem.getTitle(), Toast.LENGTH_LONG).show();
        return true;
    }
});
```

Crea una app con el código necesario para crear un NavigationView con el aspecto de la siguiente imagen (sigue los pasos de los apuntes). Cuando pulses uno de los ítems del menú deberá mostrarse el texto del ítem pulsado en un Toast. El menú Navigation podrá activarse arrastrando lateralmente o con el icono home.



Definición de Tabs en la Action Bar.

El uso de pestañas o tabs como elemento selector de distintas vistas no es algo exclusivo de Android. El elemento TabLayout de la librería de diseño simplifica el proceso de añadir pestañas en nuestra aplicación. Pueden existir más pestañas que las visibles en la pantalla y ser accesibles mediante desplazamientos izquierda o derecha sobre la pantalla.

El TabLayout se combina con el elemento ViewPager para permitir la paginación horizontal de las pestañas y el contenido de cada una de ellas.

Pueden colocarse independientemente de la Toolbar, es decir, fuera de ella, aunque nosotros en nuestro ejemplo vamos a incluir las pestañas dentro de la misma (le aplicaremos un efecto de scroll).

```
<android.support.v4.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="false">
    <android.support.design.widget.CoordinatorLayout
        android:id="@+id/coordinator"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.design.widget.AppBarLayout
            android:id="@+id/app_bar_layout"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:theme="@style/ThemeOverlay.AppCompat.Dark">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                android:background="?attr/colorPrimary"
                app:layout_scrollFlags="scroll|enterAlways"/>

            <android.support.design.widget.TabLayout
                android:id="@+id/tablayout"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:background="?attr/colorPrimary"
                app:tabGravity="fill"
                app:layout_behavior="@string/appbar_scrolling_view_behavior"/>
        </android.support.design.widget.AppBarLayout>
```

```
        <android.support.v4.view.ViewPager
            android:id="@+id/viewpager"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_behavior="@string/appbar_scrolling_view_behavior"/>

    </android.support.design.widget.CoordinatorLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/navigation_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true"
        app:headerLayout="@layout/drawer_header"
        app:menu="@menu/drawer" />
</android.support.v4.widget.DrawerLayout>
```

Una vez hecho esto, hay varios atributos importantes que podemos configurar para ajustar la apariencia de nuestra TabLayout:

- tabMode - Establece el modo de utilización del TabLayout. Puede ser fixed (todas las pestañas se muestran al mismo tiempo) o scrollable (mostrar un subconjunto de pestañas, siendo posible hacer scroll entre ellas).
- tabGravity - Establece el posicionamiento de las pestañas, que pueden ser o bien fill (distribuir todo el espacio disponible entre las fichas individuales) o el centre (pestañas centradas sobre el TabLayout).

- setText () - Este método se utiliza para establecer el texto que se mostrará en la pestaña.
- setIcon () - Este método se utiliza para establecer el icono que se mostrará en la pestaña.

Lo importante de la estructura es que el ViewPager tiene la propiedad

```
app:layout_behavior="@string/appbar_scrolling_view_behavior"
```

Lo que permite que otros elementos marcados como scroll reaccionen a los desplazamientos de esta vista, en nuestro ejemplo la Toolbar.

Las banderas de scroll controlan el tipo de comportamiento con que desaparece la Toolbar:

- scroll: indica que un view desaparecerá al desplazar el contenido.
- enterAlways:vuelve visible al view ante cualquier signo de scrolling.
- enterAlwaysCollapsed: vuelve visible el view solo si se mantiene el scroll en la parte superior del contenido.
- exitUntilCollapsed: desaparece el view hasta que sus dimensiones superen la altura mínima.

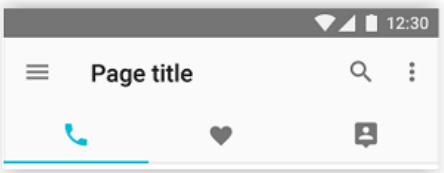
El tratamiento del TabLayout es como el de cualquier otro widget. Así que es posible obtener su instancia programáticamente con findViewById.

```
tabs = (TabLayout) findViewById(R.id.tablayout);
```

Para añadir pestañas al TabLayout utilizamos el método addTab(), que recibe como parámetro un nuevo objeto de tipo TabLayout.Tab

```
//añadimos elementos a el tab
tabs.addTab(tabs.newTab().setText("TELÉFONOS"));
tabs.addTab(tabs.newTab().setText("TABLETS"));
tabs.addTab(tabs.newTab().setText("PORTÁTILES"));
```

Un ejemplo de un TabLayout con iconos podría ser:



Con las siguientes líneas en el código:

```
...
tabs.addTab(tabs.newTab().setIcon(R.drawable.ic_teléfono));
tabs.addTab(tabs.newTab().setIcon(R.drawable.ic_tablet));
tabs.addTab(tabs.newTab().setIcon(R.drawable.ic_portátil));
```

¿Qué elementos necesitamos para visualizar el contenido de cada una de las pestañas en el ViewPager definido? ¿Cómo vamos a gestionar la selección de cada pestaña?

Para gestionar estos elementos de la forma más eficiente posible necesitamos utilizar Fragments.

Para ello crearemos una nueva clase llamada PageAdapter que hereda de FragmebtStatePageAdapter y que nos devuelve el fragment asociado a cada pestaña.

Como ya sabemos, un fragment viene definido por un archivo XML con la interfaz gráfica asociada al mismo, y por un archivo .java donde se recoge la funcionalidad del mismo.

Veamos la clase PageAdapter:

```
public class PagerAdapter extends FragmentStatePagerAdapter {
    int mNumOfTabs;

    public PagerAdapter(FragmentManager fm, int NumOfTabs) {
        super(fm);
        this.mNumOfTabs = NumOfTabs;
    }

    @Override
    public Fragment getItem(int position) {

        switch (position) {
            case 0:
                TabFragment1 tab1 = new TabFragment1();
                return tab1;
            case 1:
                TabFragment2 tab2 = new TabFragment2();
                return tab2;
            case 2:
                TabFragment3 tab3 = new TabFragment3();
                return tab3;
            default:
                return null;
        }
    }

    @Override
    public int getCount() { return mNumOfTabs; }
}
```

Realmente el adaptador a través del método getItem devuelve un objeto de tipo fragment. Tendremos un fragment por cada pestaña de nuestro TabLayout.

Posteriormente en la activity que contenga el ViewPager, asociaremos a nuestro ViewPager el adaptador creado previamente (este adaptador recibe como parámetro un objeto FragmentManger y el número de pestañas de mi TabLayout).

Veamos el código de la activity principal:

```
package com.example.director.designdemo;

import ...

public class MainActivity extends AppCompatActivity {
    DrawerLayout mDrawerLayout;
    TabLayout tabs;
    ViewPager mViewPager;
    TextView texto;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);

        setSupportActionBar(toolbar);
        ActionBar actionBar = getSupportActionBar();
        actionBar.setHomeAsUpIndicator(R.drawable.ic_menu);
        actionBar.setDisplayHomeAsUpEnabled(true);

        tabs = (TabLayout) findViewById(R.id.tablayout);
        //añadimos elementos a el tab
        tabs.addTab(tabs.newTab().setText("TELÉFONOS"));
        tabs.addTab(tabs.newTab().setText("TABLETS"));
        tabs.addTab(tabs.newTab().setText("PORTÁTILES"));
```

Para poder movernos entre los tabs mediante deslizamiento horizonta, se necesitará implementar el listener que gestione la selección de cada una de ellas TabLayoutOnPageChangeListener().

```
final ViewPager mviewPager = (ViewPager) findViewById(R.id.viewpager);
final PagerAdapter adapter = new PagerAdapter
        (getSupportFragmentManager(), tabs.getTabCount());
mviewPager.setAdapter(adapter);
mviewPager.addOnPageChangeListener(new TabLayout.TabLayoutOnPageChangeListener(tabs));
```

Pero si lo que queremos es que funcione correctamente la pulsación de la pestaña, deberemos sobreescribir el método onTabSelected del listener OnTabSelectedListener, como se muestra en la imagen:

```
tabs.setOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
    @Override
    public void onTabSelected(TabLayout.Tab tab) {
        mviewPager.setCurrentItem(tab.getPosition());
    }
    @Override
    public void onTabUnselected(TabLayout.Tab tab) { }

    @Override
    public void onTabReselected(TabLayout.Tab tab) {}
});
```

Ejercicio PropuestoTabYNavigationView